前言

本書的主題是 Java 程式語言與環境，不論是軟體開發人員或是一般的使用者一定都聽過 Java，Java 的出現帶來了 web 史上最令人興奮的成長，Java 應用程式促成了許多網際網路業務的成長。Java 可說是世界上最受歡迎的程式語言，數百萬個程式設計師在各種電腦資訊領域使用 Java，在開發人員的需求量上，Java 超越了 C++ 與 Visual Basic 等程式語言，更在以 web 為基礎的服務等特定開發領域成為實質上的標準。目前，大多數的大學介紹課程都使用 Java 再搭配其他程式語言，也許本書就是讀者的課本！

本書提供 Java 基礎與 API 完整的介紹，會名符其實的詳細介紹 Java 程式語言與類別函式庫、程式寫作技巧與常用慣例，也會深入介紹某些有趣的主題，並提供其他為人熟知主題的基本說明。歐萊禮的其他作品將以本書的介紹為基礎，更完整的提供 Java 在特定領域的資訊與應用。

本書儘可能的提供完整、實際又有趣的範例，而不是只是逐一介紹個別功能。雖然只是簡單的範例，卻能夠示範 Java 能夠做到的事情，本書的篇幅並不足以開發出下個「殺手級應用」，但作者們希望這些內容能夠作為讀者接下來數個小時實驗的起點，或是激發出讀者自行開發的想法。

目標讀者

本書的目標讀者是資訊專業人員、學生、技術人士以及年輕的駭客們，是寫給那些想要有 Java 實作經驗，以開發實際應用為目標的人看的。本書也能夠作為物件導向程式設計、網路與使用者介面的入門，在學習 Java 的過程中，從深入了解 Java 基礎及其 API 開始，從而學會強大且實際的軟體開發方法。

碁峰 www.gotop.com.tw

Java 乍看之下與 C 或 C++ 很像，如果讀者有這些語言的經驗的話，讀起本書會有些起步的優勢，沒有 C/C++ 經驗的讀者也不用擔心，不要太在意 Java 與 C 及 C++ 的語法相似性，Java 在許多方面的行為更接近於 Smalltalk 與 Lisp 這類的動態程式語言；有其他物件導向程式語言的經驗應該也會有些幫助，只是要改變一些觀念以及忘掉一些習慣。一般認為 Java 比 C++ 與 Smalltalk 更為簡單，如果讀者喜歡從簡要的範例與個人經驗學習，就應該會喜歡本書。

本書的最後一部分跳出 Java 本身，介紹了網頁應用程式、網頁服務與請求處理等主題，讀者應該要對瀏覽器、伺服器與文件等有基本的概念。

新發展

這版《*Java 學習手冊*》實際上是歐萊禮廣受歡迎的《*Exploring Java*》的第七版（更新並改名），每次改版作者都花了許多精力，除了加入介紹新功能的內容之外，也會完整的修訂與更新原有的內容，確保內容一致性，也會再加入這些年來在真實世界的觀點與經驗。

Java 新近版本的主要改變是 applets 的重要性降低了，最近幾個版本最明顯的改變在於不再強調 applet，這反應出近年來 applet 在建立互動網頁上逐漸失去了它的地位。相反的，對於 Java web 應用程式與 web service 的內容則大幅增加，這些也都是目前最為成熟的技術。

本書涵蓋了 Java「長期支援版本」的所有重要功能，官方名稱是 Java Standard Edition（SE）11，OpenJDK 11，但同時也包含了 Java 12、Java 13 與 Java 14 版本中的一些功能，Sun Microsystems（Oracle 之前的 Java 維護者）在多年前改變了命名規則，Sun 用 *Java 2* 代表 Java 1.2 版所引進的新功能，同時放棄了 *JDK* 改用 *SDK*。在第六次的發佈，Sun 從 Java 1.4 版跳到 Java 5.0，但保留了 JDK 這個名稱，並延續使用版號的慣例。但延續了版號慣例，之後接著的是 Java 6、Java 7 等，直到現在的 Java 14。

這些 Java 版本反應出一個偶有語言變動的程式語言，以及持續更新的 API 與函式庫。本書涵蓋這些新功能並儘可能的更新書中的範例，除了反應出當前的 Java 實作之外，也展現出現代的風格。

碁峰 www.gotop.com.tw

本版新增內容（Java 11、12、13、14）

本書第五版延續了儘可能完整更新到最新狀況的傳統，包含從 Java 11 開始（也就是長期支援版本）以及 Java 12、13 與 14 版的功能（第十三章會進一步說明新近版本 Java 包含與除外的功能）。本書第五版的新主題包含：

- 新語言特性，包含泛型下的型別推論（type inference），以及例外處理與自動資源管理語法的改進。
- 新的互動式介面（jshell），能夠即時測試簡短的程式碼。
- 提議的 switch 表示式。
- 基本 lambda 表示式。
- 更新了全書所有的範例與分析。

使用本書

本書結構大略介紹如下：

- 第一章與第二章提供 Java 概念的基本介紹，包含了讓讀者能直接開始寫 Java 程式的說明。
- 第三章討論 Java 開發的基本工具（編譯器、直譯器、jshell 以及 JAR 打包檔案）。
- 第四章與第五章先介紹程式設計基礎，接著說明 Java 程式語言本身，從基本語法開始，涵蓋類別與物件、例外、陣列、列舉、註釋（annotations）等。
- 第六章涵蓋例外、錯誤以及 Java 原生的 log 機制。
- 第七章包含了 collections 函式庫以及泛型與 Java 中的參數化型別。
- 第八章介紹了文字處理、格式化、掃描、字串工具等的核心 API 工具。
- 第九章說明語言內建的執行緒機制。
- 第十章介紹用 Swing 開發基本圖形使用者介面（graphical user interface，GUI）。
- 第十一章涵蓋 Java I/O、streams、檔案、socket、網路以及 NIO 套件。
- 第十二章包含 web 應用程式，使用了 servlet、servlet 過濾器以及 WAR 檔案，另外也介紹了 web service。

碁峰 www.gotop.com.tw

- 第十三章介紹 Java Community Process，並說明追蹤後續 JAva 版本更新的方法，能夠幫助讀者用新功能翻新現有程式碼，例如 Java 8 引進的 lambda 表示式就是很好的例子。

如果讀者跟作者們一樣，不會從頭到尾一頁一頁的讀完一本書，如果你真的像我們一樣，通常也會跳過前言，只是，萬一這次剛好看到這裡，以下是一些建議：

- 如果你已經是程式設計師，需要很快的學會 Java，可能想要找一些例子，那麼可以先翻閱一下第二章的入門介紹，要是這樣還不夠，那就至少要看一下第三章的內容，第三章會介紹編譯器與直譯器的使用方式，接著應該就可以開始動手了。
- 如果想要寫網路或 web 式應用與服務，就應該看第十一與十二章，網路仍然是 Java 最有趣也最重要的主題之一。
- 第十章介紹了 Java 的圖形特性與元件架構，對開發桌面用圖形化 Java 應用程式有興趣的讀者應該要讀這章。
- 第十三章討論了跟上 Java 語言改變的方法，這部分的討論是專屬 Java 程式語言本身，不針對特定領域。

線上資源

網路上有許多提供 Java 資訊的線上資源。

Oracle 的 Java 官網是 *https://oreil.ly/Lo8QZ*，可以在這裡找到軟體、更新與 Java 發佈版，你可以在這裡找到 JDK 的參考實作，包括編譯器、直譯器以及其他工具。

Oracle 同時也維護的 OpenJDK 網站（*https://oreil.ly/DrTm4*），這是最主要的開放源碼版 Java 及相關工具，本書範例都會使用 OpenJDK。

你也應該拜訪歐萊禮的網站 *http://oreilly.com/*，你可以在這裡找到其他歐萊禮書籍的資訊，除了 Java 外還有其他持續成長的主題，同時也應該看看線上學習與研討會等資源，歐萊禮在教育方面非常的傑出。

當然，你還應該看看 Java 學習手冊的官網（*http://oreily.ly/Java_5E*）。

碁峯 www.gotop.com.tw

讀者應該可以看到如圖 2-12 的畫面，恭喜！現在已經執行了第二個 Java 程式了！可以稍微享受一下畫面上的光芒。

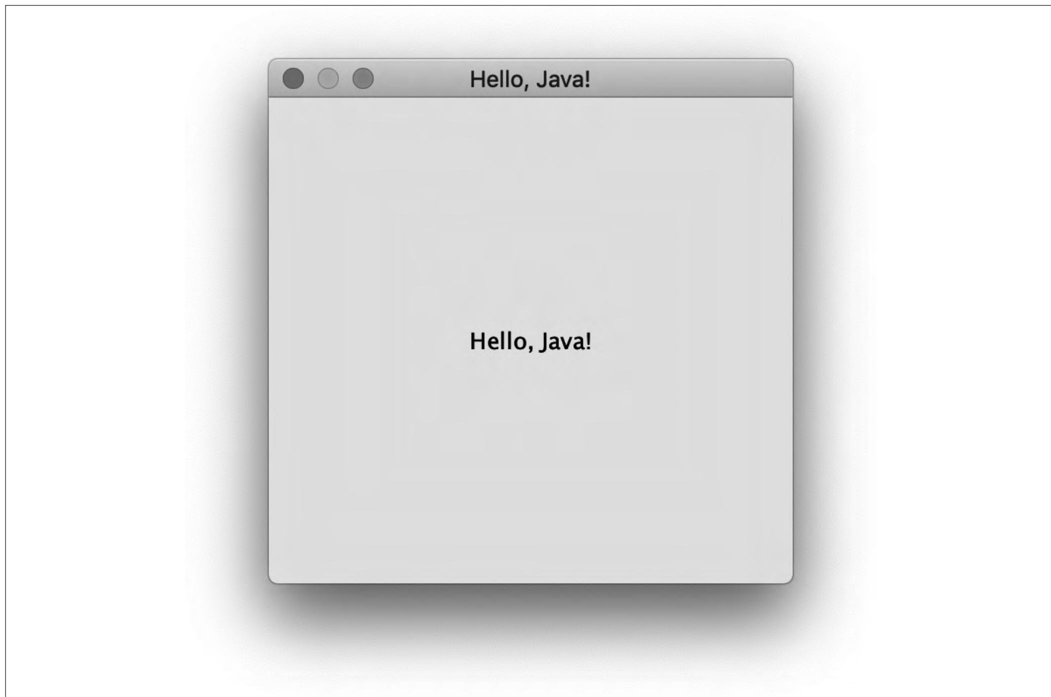

圖 2-12　HelloJava 程式的輸出結果

需要注意的是，按下關閉視窗的按鈕後雖然視窗會消失，但程式仍會繼續執行（之後的版本會修正關閉行為）。要在 IDEA 中停止 Java 程式，可以點選先前執行程式用的綠色執行按鈕右側的紅色方框鈕。如果是從命令列執行範例，就按下鍵盤上的 Ctrl-C 組合鍵，當然也可以一次執行多個程式（多個執行實例）。

`HelloJava` 雖然是個小程式，但背後包含許多的行為，這幾行簡短的程式碼只是冰山一角，表層之下的是由 Java 程式語言與 Swing 函式庫所提供的層層功能。要記得本章我們只是很快的帶過一次，讓讀者知道全貌，本章雖然試著提供足以理解的細節，但更深入的說明會留後適當的章節。這種做法同時適用於 Java 程式語言以及物件導向觀念，說到這裡，接下來讓我們看看這個範例程式到底做了些什麼。

碁峰 www.gotop.com.tw

類別

第一個例子定義了名為 HelloJava 的 class：

```
public class HelloJava {
...
```

類別（class）是大多數物件導向程式語言的基本區塊。**類別**是資料以及操作資料的函式的組合，類別裡的資料稱為**變數**（*variables*），有時也稱為**欄位**（*fields*）。在 Java 裡，函式（function）一般稱為**方法**（*method*）。物件導向式語言主要的好處就是類別裡資料與功能的關連性，以及類別能夠**封裝**（*encapsulate*）或隱藏資料，讓開發人員不需要擔心低階細節。

在應用程式裡，類別代表一些實際的東西，例如畫面上的按鈕或試算表裡的資訊，也可以代表一些抽象事物，如排序演算法或是遊戲角色的疲勞度；代表試算表的類別可以有表示每個欄位數值的變數，以及操作欄位的方法，例如「清除整列」（clear a row）或「計算數值」（compute value）。

範例中的 HelloJava 類別是一個完整 Java 應用程式的單一類別，只定義了一個 main() 方法，方法的內容就是程式的主體：

```
public class HelloJava {
  public static void main( String[] args ) {
  ...
```

啟動應用程式時，首先會執行的就是這個 main() 方法，其中標記著 String[] args 的部分能夠傳入**命令列參數**（*command-line argument*）給應用程式使用。下一節會逐行說明 main() 方法的內容。最後，雖然這個版本的 HelloJava 沒有定義任何屬於 class 的變數，但它的確在 main() 方法裡使用了兩個變數——frame 跟 label，我們很快就會再介紹這兩個變數。

main() 方法

先前我們在執行範例時看到，執行 Java 應用程式表示指定所需功能的類別，將類別名稱稱作為參數傳給 Java 虛擬機器。接著 java 命令會尋找 HelloJava 類別，確認是否包含名為 main() 的特殊方法，並檢查方法擁有正確的格式。如果一切都符合的話，就會開始執行；要是找不到執行的標的，就會得到錯誤訊息。main() 方法是應用程式的進入點，每個獨立的 Java 應用程式至少都會有一個含有 main() 方法的類別，能夠執行啟動程式其他部分的工作。

碁峰 www.gotop.com.tw

範例裡的 main() 方法設定了一個視窗（一個 JFrame）負責 HelloJava class 的視覺輸出，目前，這個方法負責應用程式的一切工作，但在物件導向式程式裡，一般會將責任委派（delegate）給許多不同的類別。在範例的下一階段變化裡，會開始做這樣的切割（建立第二個 class），隨著範例程式持續成長，我們會看到 main() 方法大體上仍然維持相同的樣貌，只是負責啟動程序。

接下來簡單的看過 main() 方法，確定知道每一個步驟在做些什麼。首先，main() 建立了一個 JFrame，一個代表範例的視窗：

```
JFrame frame = new JFrame("Hello, Java!");
```

這行程式碼裡的 new 十分重要，JFrame 是類別名稱，這個類別代表了螢幕上的一個視窗，但類別本身只是個樣板，就像是建築規劃而已；new 關鍵字告訴 Java 配置記憶體，並真正建立一個特定的 JFrame 物件。以這個例子來說，在 JFrame 括號裡的參數，告訴了 JFrame 該在標題列顯示的訊息，也可以去掉「Hello, Java」字串，只用空括號建立一個沒有標題訊息的 JFrame，但這只是因為 JFrame 特別允許這樣的做法。

JFrame 視窗一開始建立的時候非常的小，在顯示 JFrame 之前，必須將它設為適當的大小：

```
frame.setSize( 300, 300 );
```

這是另一個呼叫特定物件方法的例子，這個例子裡，setSize() 方法是由 JFrame class 所定義，能夠影響以 frame 變數儲存的這個 JFrame 物件。與 frame 相同的，範例裡也建立了一個 JLabel 的實體（instance）用來存放視窗裡的文字：

```
JLabel label = new JLabel("Hello, Java!", JLabel.CENTER );
```

JLabel 就像是個真正的標籤，能夠在特定位置放置文字（範例程式是在視窗裡），這是非常物件導向式的概念：讓物件持有文字，而不是直接使用「draw」（描繪）之類的方法畫出文字再繼續下一步。這背後的理由會愈來愈清楚。

接著，必須把標籤放進先前建立好的視窗裡：

```
frame.add( label );
```

這裡呼叫了 add() 方法將 label 放進 JFrame 裡，JFrame 是個能夠持有其他東西的容器（container），稍後會再討論這點。main() 的最後一項工作是顯示視窗與它的內容，否則使用者就看不到任何東西。一個看不見的視窗會讓程式非常無趣。

```
frame.setVisible( true );
```

碁峰 www.gotop.com.tw

這就是整個 main() 方法了，雖然 HelloJava 類別隨著一路下來的範例有所變化，這些變化都是圍繞著 main() 方法，方法本身大體上並沒有太大的改變。

類別與物件

類別（class）是應用程式某個部分的藍圖，擁有構成該元件的方法與變數。在應用程式活動期間，每個類別可以同時有許多個別不同的運作複本，這些獨立的運作複本被稱為類別的**實體**（*instance*），也稱為**物件**（*object*）。同一個類別的兩個實體可以有不同的資料，但永遠會有相同的方法。

以 Button 類別為例，只會有一個 Button 類別，但應用程式可以建立許多不同的 Button 物件，每個都是同一個類別的實體。此外，兩個 Button 實體可以擁有不同的資料，也許是有不同的外觀與執行不同的動作，就這個意義上，類別可以看成是許多物件的模子，類似餅乾模型切割刀，在電腦記憶體中產生出自己的運作實體。稍後會看到還不只如此（實際上類別可以與產生的實體共享資料），但目前這樣的解釋就夠了。第五章對類別與物件有完整的介紹。

物件（*object*）是非常常見的詞，在某些情境下跟**類別**可以互換使用，物件是所有物件導向式程式語言以某些型式指涉的抽象實體（abstract entity），我們會用**物件**表示類別的實體。也就是說，可能會把 Button 類別產生的實體稱為按鈕（button）、Button 物件或不特別指明類別，只稱為物件。

前一個範例的 main() 方法建立了一個 JLabel 類別的實體，顯示在 JFrame 類別的實體上。讀者可以修改 main() 方法，產生多個 JLabel 實體，也許可以把每個實體放在不同的視窗裡。

變數與 Class 型別

在 Java 裡，每個類別都定義了一個新的**型別**（*type*，資料型別），變數可以宣告為該型別，持有這個類別的實體。例如，變數可以是 Button 型別並持有 Button 類別的實體，也可以是 SpreadSheetCell 型別並持有 SpreadSheetCell 物件，就如同 int 或 float 這些簡單得多的型別一樣，都表示了數字。變數具有型別，無法持有任何種類物件的限制是程式語言確保程式碼安全性與正確性的另一個重要特性。

暫時先忽略 main() 方法內部使用的變數，如此一來，整個簡單的 HelloJava 範例只宣告了一個變數，就在 main() 方法的宣告：

```
public static void main( String [] args ) {
```

碁峰 www.gotop.com.tw

如同其他程式語言裡的函式（function），Java 裡的方法宣告了能夠作為**引數**（*argument*）的**參數**（*parameter*，變數）列，同時也指定了各個參數的型別。以這個情形來說，main 方法要求在被呼叫的時候，必須傳入 String 物件的陣列到名為 args 的變數，String 是 Java 裡表示文字的基本物件。先前提到過 Java 使用 args 參數傳遞任何指定給 Java 虛擬機器（VM）的命令列引數到應用程式裡（目前還沒有用到這部分）。

到目前為止，都只是很不嚴謹的說變數會持有物件，實際上，具有類別型別的變數並沒有包含物件太多部分，而是指向它們。類別型別變數是物件的參考（reference），**參考**是指向物件的指標或者說是物件的標記（handle），如果宣告了一個類別型別的變數但沒有指派任何物件，它就不會指向任何東西。會指派為預設值 null，表示「沒有值」，如果把指向 null 值的變數當作有指向真實物件的變數一般的使用，就會發生執行期錯誤 NullPointerException。

當然，物件的參考總是得來自於某個地方，在範例裡是透過 new 運算子建立了兩個物件，本章稍後會再更深入的討論物件建立的過程。

HelloComponent

到目前為止，我們的 HelloJava 範例只由單一個類別組成，實際上，由於行為十分簡單，它其實也只需要一個比較大的方法就夠了，雖然已經使用了幾個物件顯示 GUI 訊息，但我們的程式碼並沒有呈現任何物件導向式結構。為了修正這個問題，接下來我們馬上會加入第二個類別，作為本章後續發展的基礎，接下來的程式會取代 JLabel 類別的工作（再見，JLabel！），更換為自行開發的圖形介面類別：HelloComponent。HelloComponent 一開始十分簡單，只是在固定的位置顯示「Hello, Java!」訊息，稍後會再增加其他的能力。

新類別的程式碼十分簡單，只多加了幾行：

```
import java.awt.*;

class HelloComponent extends JComponent {
  public void paintComponent( Graphics g ) {
    g.drawString( "Hello, Java!", 125, 95 );
  }
}
```

碁峰 www.gotop.com.tw

可以直接把這段文字加進 *HelloJava.java* 檔案裡，也可以另外建立一個 *HelloComponent.java* 檔。如果放在同一個檔案裡，就必須把 import 指令移到檔案的最上方，與其他的 import 放在一起；要用這個新建立的類別取代 JLabel，只需要把那兩行表示標籤的程式碼改成：

```
frame.add( new HelloComponent() );
```

這次編譯 *HelloJava.java* 的時候，會看到產生兩個二進制類別檔：*HelloJava.class* 與 *HelloComponent.class*（不論原始程式碼採用何種型式）。程式碼執行的效果與 JLabel 版本十分相近，但一旦你調整視窗大小，就會發現新的程式不會自動調整位置，讓文字保持置中。

為什麼要做這些變動？為什麼改動程式碼之後卻破壞了原先有著完美行為的 JLabel 元件呢？我們建立了新的 HelloComponent 類別，**擴展**通用的 JComponent 圖形類別，擴展（extend）類別就表示要在原有的類別上增加新的功能，建立新的類別，這在下一節會深入說明。現在我們建立了新的 JComponent，擁有稱為 paintComponent() 的方法，這個方法負責畫出訊息內容。paintComponent() 方法需要一個名為 g 的引數（名稱有點太簡單了），型別則是 Graphics，呼叫 paintComponent() 時，會將一個 Graphics 物件指派給 g，讓程式在方法主體中使用；我們稍後會更詳細討論 paintComponent() 與 Graphics 類別，至於為什麼要這麼做，等到為這個類別加上其他功能之後，你就會了解了。

繼承

Java 的類別是以親 - 子階層的關係安排，其中的親代與子代分別也稱為 *superclass* 與 *subclass*，在第五章會更詳細說明這些觀念。在 Java 裡，每個類別都會有一個，也只能有一個上層類別（superclass，單一親代），但可能會有許多子類別（subclass），唯一例外是 Object 類別，Object 位於整個類別階層的最上方，沒有上層類別。

在先前的範例中宣告我們新增加的類別時，使用了 extends 關鍵字表示 HelloComponent 是 JComponent 類別的子類別：

```
public class HelloComponent extends JComponent { ... }
```

子類別可能會繼承上層類別的部分或所有的變數與方法，透過繼承，子類別可以像自己所宣告的變數與方法一般的使用來自上層類別的變數與方法，也可以自行增加新的變數與方法，還可以**覆寫**（*override*）或改變繼承而來的方法的意義。使用子類別的時候，被覆寫的方法會被子類別所提供的版本取代，繼承透過這種方式提供了一種強大的機制，能讓子類別改變或延伸來自上層類別的功能。

碁峰 www.gotop.com.tw

例如，為了產生新的科學試算表類別，可能會從先前的試算表類別建立子類別，加上數學計算函式以及其他內建常數。在這種情況下，科學試算表的原始碼裡可能會宣告一些新加入的數學函式與代表特殊常數的值，這個新類別會自動擁有從代試算表繼承而來，基礎試算表類別所具有的變數與方法，這也表示科學試算表將自己視為試算表，程式設計師可以將擴充版本試算表用在所有簡單版試算表可以使用的地方，最後這句話有著十分重要的影響，也是本書會持續探討的主題，這句話的意義表示特殊化的物件可以被視為更一般性的物件使用，調整它們的行為卻不改變底層的應用，這就稱為「多型」（polymorphism），是物件導向式程式設計的基礎。

HelloComponent 是 JComponent 類別的子類別，繼承了許多沒有顯示在範例程式碼裡的變數與方法，這些繼承來的方法正是這小小的類別能夠作為 JFrame 元件的原因，我們只加上了一點點的調整。

JComponent 類別

JComponent 提供了建立各種型式 UI 元件所需的基本框架，按鈕、標記與下拉列表等特殊的元件都是實作為 JComponent 的子類別。

我們會覆寫子類別中的方法，實作特殊元件的行為，被限制在某些預先定義好的程序聽起來似乎有很多限制，實際上並非如此。要記得這些方法都是與視窗系統互動的方式，不需要把整個應用程式的邏輯塞到裡面。實際上的應用程式可能會包含上百、上千個類別，有難以計數的方法與變數，還同時執行很多個執行緒，其中大多數都與工作的特定部分有關（這些稱為**業務物件**（*domain* object））。JComponent 及其他預先定義好的類別只是提供框架，負責處理特定類型的使用者介面或是顯示資訊給使用者。

paintComponent() 是 JComponent 類別十分重要的方法，透過覆寫的方式，能夠以不同的方式實作元件顯示在螢幕的樣貌，paintComponent() 預設的行為並不會在螢幕上畫出任何東西，如果子類別不覆寫這個方法，就會以透明的方式呈現元件。範例程式只對 paintComponent() 作稍稍有趣的覆寫，對其他繼承自 JComponent 的成員則不作任何覆寫，因為其他成員提供的是基本功能，對目前這個簡單的範例而言都是合理的預設行為。隨著 HelloJava 的變化，我們會繼續深入研究其他繼承而來的成員，使用更多的方法，同時 HelloComponent 也會為了自行獨特的需要，加入一些應用程式專屬的方法與變數。

JComponent 實際上也只是名為 Swing 的冰山上的一角，Swing 是 Java 的 UI 工具庫，在範例程式中是以最上方的 import 指令的型式呈現，我們在第十章會作更深入的討論。

碁峰 www.gotop.com.tw

關係與指涉

可以把 HelloComponent 視為是 JComponent 的原因在於，建立子類別的過程可以想成是建立「is a」（是個）的關係，也就是子類別「is a」上層類別，因此 HelloComponent 就是一種 JComponent。在程式中參照某種物件，其意義是該物件的類別及其子類別的任何實體，稍後我們在更詳細檢視 Java 類別階層時，會看到 JComponent 本身是 Container 類別的子類別，而 Container 則又衍生自 Component 類別等等，如圖 2-13。

從這個角度來看，一個 HelloComponent 物件就是個 JComponent，也就是個 Container，這些最終都能夠被視為是個 Component。HelloComponent 就是透過這些類別繼承了基本的 GUI 功能，以及（稍後會看到）能夠嵌入其他圖形元件的能力。

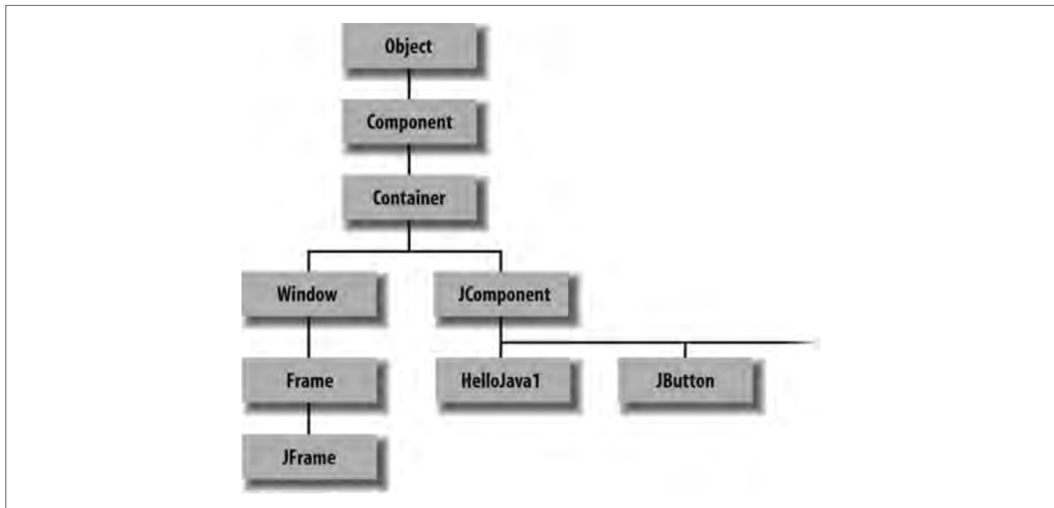

圖 2-13　Java 結構的一部分

Component 是最上層 Object 類別的子類別，所以先前提到的這些類別全都是屬於 Object 類型的類別，Java API 裡的所有類別都繼承了 Object 的行為，也就是在第五章會看到的一些基本行為。我們接下來仍然會用比較通用的物件表示任何類別產生的實體，當要表示特定型別的類別時則會使用 Object。

碁峰 www.gotop.com.tw

套件與匯入

先前提到過範例程式的第一行程式碼能讓 Java 知道使用類別的所在位置：

```
import javax.swing.*;
```

更明確的來說，這行程式碼告訴編譯器要使用 Swing GUI 工具箱裡的類別（對先前的範例來說就是 `JFrame`、`JLabel` 以及 `JComponent`），這些類別放在名為 `javax.swing` 的 Java *package*（套件），在 Java 套件指的是有相同目的應用的類別形成的集合，同一個套件裡的類別彼此之間會比其他類別有更高的存取權限，能夠設計成更緊密的合作關係。

套件是以 . 分隔的階層式命名，例如 `java.util` 與 `java.util.zip`，套件裡的類別必須遵守它們在 classpath 中的規則，它們的「全名」裡也必須包含套件名稱，以更適當的術語來說應該稱為**完全限定名稱**（*fully qualified name*），例如 `JComponent` 的完全限定名稱是 `javax.swing.JComponent`，可以直接使用完全限定名稱取代 `import` 指令：

```
public class HelloComponent extends javax.swing.JComponent {...}
```

`import javax.swing.*` 這行指令能讓程式透過簡單的名稱使用 `javax.swing` 套件裡的所有類別，就不需要用完全限定名稱表示 `JComponent`、`JLabel` 與 `JFrame` class 了。

在第二個範例加入額外的類別時，可以看到一個 Java 原始碼檔案裡可以有不止一個 `import` 指令。`import` 的作用像是建立「搜尋路徑」，讓 Java 知道看到非完全限定名稱時，該到哪些地方尋找這些只使用了簡單名稱的類別（實際上並不是路徑，但這種說法可以避免一些可能造成錯誤的名稱），先前看到的 `import` 使用了 `.*` 的型式表示應該要匯入整個套件，但讀者也可以只指定單一個類別，例如這個範例在 `java.awt` 套件中只用到了 `Graphics` 類別，就可以使用 `import java.awt.Graphics`，而不是透過萬用字元（*）引入 Abstract Window Toolkit（AWT）套件裡所有的類別，但我們計畫稍後會使用這個 package 裡其他的幾個類別。

`java.` 與 `javax.` 是特別的套件階層，所有以 `java.` 開頭的套件都屬於 Java 核心 API，在所有支援 Java 的平台上都可以使用，而 `javax.` 套件一般則表示對核心平台的標準擴充，不一定會安裝。但在最近幾年，有許多標準擴充被納入 Java 核心 API 後並沒有重新命名，如 `javax.swing` 就是個例子，這些套件名稱雖然以 `javax.` 開頭，但已經屬於核心 API 的一部分。圖 2-14 畫出了一些 Java 核心套件與其中的一、兩個類別。

碁峰 www.gotop.com.tw

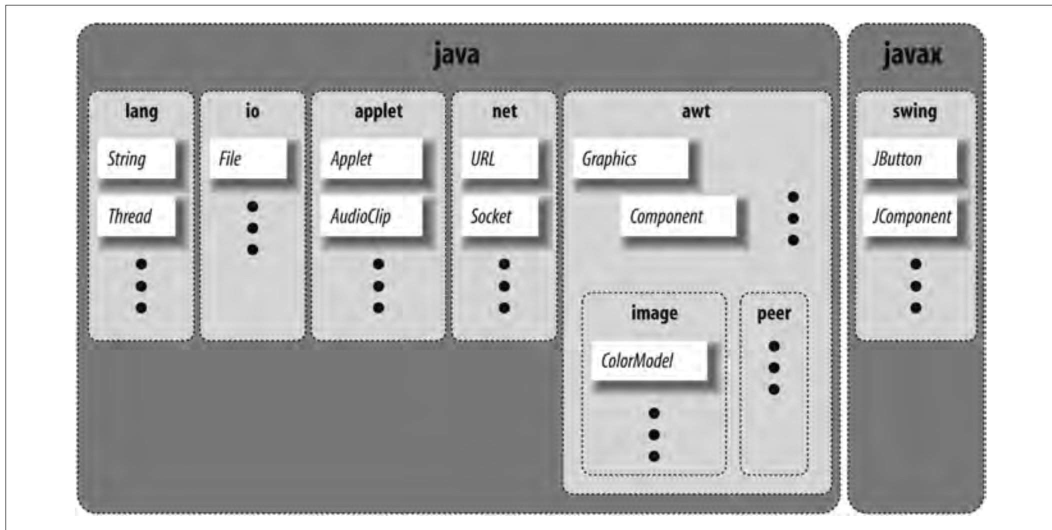

圖 2-14　一些核心 Java 套件

java.lang 包含了 Java 語言本身需要的基本類別，會自動匯入套件內的類別，程式中不需要任何 import 命令就能夠使用 String 與 System 等類別名稱，java.awt 套件中的是比較老舊的圖形化 AWT 類別，java.net 則是網路相關的類別，依此類推。

隨著對 Java 愈來愈熟悉，就能夠慢慢掌握各個套件中包含的類別及其作用，能夠正確使用這些類別是成為成功 Java 開發人員的關鍵。

paintComponent() 方法

HelloComponent 類別的原始碼裡定義了 paintComponent() 方法，覆寫了 JComponent 當中的 paintComponent() 方法：

```
public void paintComponent( Graphics g ) {
    g.drawString( "Hello, Java!", 125, 95 );
}
```

paintComponent() 被呼叫的時機是範例程式需要在螢幕上畫出自己的時候，這個方法只接受一個參數，一個 Graphics 物件，也不會傳回任何型別的傳回值（void）給呼叫者。

碁峰
www.gotop.com.tw

修飾子（*modifier*）是指放在類別、變數與方法宣告之前的關鍵字，能夠改變它們的可存取範圍、行為與語義，paintComponent() 宣告為 public，表示能夠被 HelloComponent 之外 class 的方法呼叫。以這個例子來說，就是 Java 的視窗環境會呼叫範例中的 paintComponent() 方法，而宣告為 private 的方法或變數則只能被類別自身使用。

Graphics 物件是個 Graphics 類別的實體，代表了特定的圖形化描繪區域（也稱為 *graphics context*），其中包含了能夠用來在區域裡描繪的方法，傳入到 paintComponent() 方法裡的 Graphics 物件會對應到 HelloComponent 在螢幕中視窗裡的對應區域。

Graphics 類別提供了渲染形狀、影像與文字相關的方法，在 HelloComponent 裡呼叫的是 Graphics 物件的 drawString() 方法，會在指定的坐標畫出訊息。

在先前的程式裡可以看到，呼叫特定物件方法的方式是在物件名稱後先加上 . 再加上方法名稱，所以呼叫 Graphics 物件（透過 g 變數參照）的 drawString() 方法的方式如下：

```
g.drawString( "Hello, Java!", 125, 95 );
```

讀者可能需要一些時間才能夠接受這樣的概念，應用程式的內容區域是由其他系統在任意時間呼叫的方法所畫出來的，這種情況該怎麼做出有用的事？該怎麼控制該完成什麼與什麼時候完成？稍後會回答這些問題，目前，只需要先考慮該如何以回應命令的方式組織程式，而非自行主動的結構。

HelloJava2：續集

完成了基本型之後，接下來要增加應用程式的互動能力，以下的小改版能讓使用者用滑鼠拉動文字。如果讀者是程式設計新手，那麼這次的改版也許沒那麼小，不要害怕！在往後的章節會詳細介紹這個範例所涵蓋的所有主題，目前請先享受把玩範例程式，如果不習慣範例裡的程式碼，也可以利用這個機會好好習慣建立與執行應用程式的過程。

為了避免讀者因為繼續擴展先前的範例而有所混淆，接下來的範例會稱為 HelloJava2，但接下來的改變都集中在增加 HelloComponent 的能力，並在每次改動的同時改變類別名稱，讓改動與類別名稱對應（如 HelloComponent2、HelloComponent3 等等）。由於先前介紹了繼承，也許會有讀者好奇怎麼不建立 HelloComponent 的子類別，對於接下來的範例，利用繼承的機制在原有的基礎上增加功能並不會帶來太大的好處，從頭來過比較能夠明確呈現範例的作用。

碁峰 www.gotop.com.tw

以下是 HelloJava2 的程式碼：

```
// 檔案：HelloJava2.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class HelloJava2
{
  public static void main( String[] args ) {
    JFrame frame = new JFrame( "HelloJava2" );
    frame.add( new HelloComponent2("Hello, Java!") );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setSize( 300, 300 );
    frame.setVisible( true );
  }
}

class HelloComponent2 extends JComponent
    implements MouseMotionListener
{
  String theMessage;
  int messageX = 125, messageY = 95; // 訊息的坐標

  public HelloComponent2( String message ) {
    theMessage = message;
    addMouseMotionListener(this);
  }

  public void paintComponent( Graphics g ) {
    g.drawString( theMessage, messageX, messageY );
  }

  public void mouseDragged(MouseEvent e) {
    // 儲存滑鼠坐標並畫出訊息
    messageX = e.getX();
    messageY = e.getY();
    repaint();
  }

  public void mouseMoved(MouseEvent e) { }
}
```

雙斜線表示該行後面是註解，為了讓讀者了解程式碼的變化，我們在範例程式中加上了一些註解。

碁峰 www.gotop.com.tw

將範例中的文字以 *HelloJava2.java* 為名儲存為檔案，再依先前介紹的方式編譯，應該就會得到新的 .class 檔案：*HelloJava2.class* 與 *HelloComponent2.class*。

使用以下命令執行範例：

```
C:\> java HelloJava2
```

如果讀者繼續使用 IDEA，可以點擊執行鈕執行，可以自行將顯示的「Hello, Java!」改成任何文字，好好玩一下，用滑鼠拉動文字的位置。另外要注意的是，點選視窗關閉鈕時會結束程式，其原理稍後在討論到事件（event）時會再作討論，現在先看看作了哪些變動。

實體變數

新版範例在 HelloComponent2 class 增加了一些變數：

```
int messageX = 125, messageY = 95;
String theMessage;
```

messageX 與 messageY 是整數，存放了訊息的當前坐標，程式粗暴的以固定的預設值初始化坐標，也就是接近畫面中心的位置。Java 的整數是 32 位元的有號數，可以很輕易的存放坐標值。theMessage 變數是 String 型別，能夠存放 String 類別的實體。

讀者應該注意到了，這三個變數是宣告在類別定義的大括號之內，但並不在任何方法的大括號當中，這些變數稱為**實體變數**（*instance variable*），屬於物件本身的一部分，更精確的來說，這個類別的每個個別的實體裡都會有這些變數的副本，類別內的所有方法都可以看得到（與使用）相同類別的實體變數，依據修飾子的不同，實體變數也可能被其他類別存取。

除非特別初始化實體變數，否則都會是依型別而定的預設值 0、false 或 null 等。數值型別的預設值是 0，布林型別的預設值是 false，而類別型別變數的預設值都是 null，也就表示「沒有數值」，使用 null 值的物件會造成執行期錯誤。

實體變數相對於方法引數以及其他宣告在特定方法生命週期內的變數不同，後者稱為**區域變數**（*local variable*），它們實際上是只能被方法或特定程式碼區域內部看到的私有變數，如果沒有指派數值就直接使用區域變數，編譯器會產生編譯期錯誤。區域變數的生命週期就是方法的執行期間，除非另外儲存變數的數值，否則會隨著方法結束而消失，每次呼叫方法時，都會重新建立區域變數也必須要指派數值。

碁峯 www.gotop.com.tw

透過新的變數能賦予不靈活的 paintComponent() 更多的動態，現在呼叫 drawString() 時，所有的引數都決定於這些變數。

建構子

HelloComponent2 類別包含了一個特殊的方法，稱為**建構子**（*constructor*），每次建立新實體時都會呼叫對應類別的建構子。建立新物件時，Java 會配置物件的儲存空間，將實體變數設定為預設值，接著呼叫類別的建構子方法執行應用程式層級需要的相關設定。

建構子的名稱一定要跟類別名稱相同，例如 HelloComponent2 類別的建構子就稱為 HelloComponent2()。建構子沒有傳回值，但可以想成是建立所屬類別型別的物件，建構子與其他方法一樣可以有引數，它們的終身職志就是設定與初始化新產生出來的類別實體，也許會藉助於參數傳入的資訊。

物件是使用 new 運算子再加上類別建構子及所有需要的引數建立，產生的物件會以數值傳回，在範例中，main() 方法中的新 HelloComponent2 實體是在以下這行程式建立：

```
frame.add( new HelloComponent2("Hello, Java!") );
```

這行程式碼實際上做了兩件事，為了方便理解，可以改寫成以下兩行的型式：

```
HelloComponent2 newObject = new HelloComponent2("Hello, Java!");
frame.add( newObject );
```

第一行程式是重點，也就是產生新 HelloComponent2 物件，HelloComponent2 建構子需要一個字串作為引數，依據在程式碼中的設定，這個引數會成為顯示在視窗裡的訊息內容，Java 編譯器會將 Java 程式碼中以雙引號括起來的文字轉換為 String 物件（參看第八章對 String 類別的討論），第二行只是將新元件加入視窗，呈現在使用者，這部分與前一個範例程式相同。

既然談到了設定顯示訊息，如果讀者想要更有彈性的設定訊息，可以將建構子所在的程式碼改成：

```
HelloComponent2 newobj = new HelloComponent2( args[0] );
```

如此就可以像以下的方式，在執行應用程式的時候，從命令列參數指定顯示的訊息：

```
C:\> java HelloJava2 "Hello, Java!"
```

碁峯 www.gotop.com.tw

args[0] 代表命令列的第一個參數，在第四章討論到陣列的時候，讀者就能夠更理解這個表示方式代表的意義，如果是使用 IDE 的話，需要在執行之前特別設定好命令列的參數，如圖 2-15 顯示的 IntelliJ IDEA 的設定畫面。

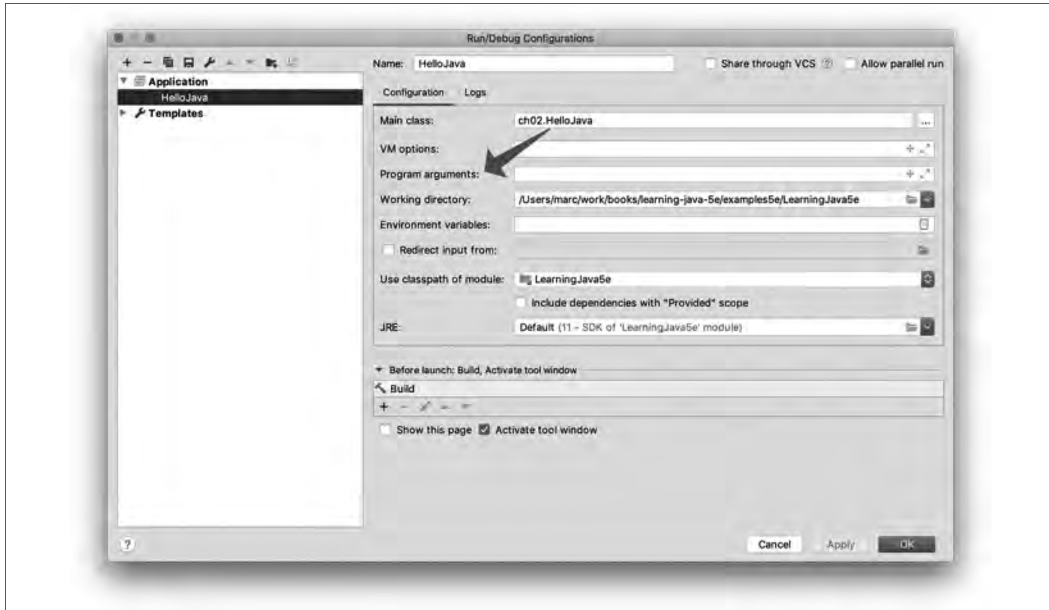

圖 2-15　IDEA 設定命令列參數的對話框

HelloComponent2 的建構子接著作了兩件事：設定 theMessage 實體變數，接著呼叫 addMouseMotionListener()，這個方法屬於稍後會討論的事件機制，addMouseMotionListener() 會告訴系統「嘿！我對滑鼠發生的所有事情都有興趣。」

```
public HelloComponent2(String message) {
  theMessage = message;
  addMouseMotionListener( this );
}
```

呼叫 addMouseMotionListener() 時使用的 this 是個特殊、唯讀的變數，用來明確的表示對物件的參照（「目前」所在的物件）。在方法裡可以用 this 表示目前執行這個方法的物件實體，也就是說，以下的兩行程式會有相同的效果，都會指定 theMessage 實體變數的數值：

```
theMessage = message;
```

碁峯 www.gotop.com.tw

或：

```
this.theMessage = message;
```

一般使用實體變數時會使用比較簡單、不明確指明目前物件的型式，但在需要將目前物件作為參數傳給另一個類別的方法時，就需要使用 this。這種做法通常用來讓其他類別能夠呼叫我們的公開物件或使用公開變數。

事件

HelloComponent2 的最後兩個方法 mouseDragged() 與 mouseMoved() 能讓程式取得滑鼠的資訊。每次使用者執行操作時，不論是按下鍵盤上的按鍵、移動滑鼠，甚至是把頭撞上觸控螢幕上，Java 都會產生**事件**（*event*），事件代表發生了某項操作，包含該操作的相關資訊（如時間與地點）。大多數的事件都會對應到應用程式的特定 GUI 元件，例如，按下鍵盤按鍵可能代表在文字輸入欄位中輸入了某個字元，按下滑鼠按鈕可能會觸發螢幕上的特定按鈕，即使只是在螢幕的特定區域移動滑鼠都可以觸發如提醒或改變滑鼠游標形狀等效果。

為了處理這些事件，程式匯入了 java.awt.event 這個新套件，程式使用 java.awt.event 所提供的各式各樣 Event 物件取得使用者的資訊（要特別注意的是，匯入 java.awt.* 並不會自動匯入 event 套件，import 指令並不會遞迴匯入，雖然套件是以階層的方式命名，但這並不表示套件間有包含關係）。

有許多不同的事件類別，如 MouseEvent、KeyEvent 與 ActionEvent，事件的意義大都十分直覺，MouseEvent 發生表示使用者對滑鼠做了一些動作，KeyEvent 則表示使用者按了鍵盤上的按鍵等等，ActionEvent 則有所不同，我們會在第十章介紹它的作用，目前先專心處理 MouseEvent。

Java 的 GUI 元件會對使用者的操作產生不同類型的事件，例如，當使用者在元件裡按下滑鼠按鈕，元件就會產生滑鼠事件。物件可以透過對事件來源註冊**傾聽器**（*listener*）的方式要求接收來自一個以上的元件的事件；如果想要宣告傾聽器想要接收某個元件的滑鼠動作（mouse motion）事件，就必須呼叫元件的 addMouseMotionListener() 方法，指定傾聽器物件作為引數，這也就是範例程式在建構子裡的行為，在範例程式裡元件以 this 為引數呼叫自己本身的 addMouseMotionListener() 方法，表示「我想要收到我自己的滑鼠動作事件」。

碁峰 www.gotop.com.tw

這就是註冊要收到事件的方式，但要怎麼真正的得到事件呢？這就得看 class 裡那兩個與滑鼠相關的方法了。當使用者拉動滑鼠時（也就是按著任何一個滑鼠按鍵同時移動滑鼠），傾聽器會自動呼叫 mouseDragged() 方法接收產生的事件，mouseMoved() 方法則會在使用者沒有按下按鈕，在區域內移動滑鼠時被呼叫，範例中將這兩個方法都放在 HelloComponent2 裡，並註冊自己作為自己的傾聽者，這種做法完全適合於這個可以拉動文字的新元件，但一般而言，良好的設計通常會要求以**轉接器**（*adapter*）**類別**的方式實作事件傾聽器，在 GUI 與「業務邏輯」之間有更適當的分隔，我們在第十章會更詳細討論其中的細節。

mouseMoved() 方法很無趣：什麼事也沒做。範例略過一般的滑鼠移動，將注意力集中在拉動上，mouseDragged() 就比較有內容了，視窗系統會一再呼叫這個方法，提供最新的滑鼠位置，程式碼如下：

```
public void mouseDragged( MouseEvent e ) {
  messageX = e.getX();
  messageY = e.getY();
  repaint();
}
```

mouseDragged() 的第一個引數是 MouseEvent 物件 e，其中包含了對這個事件想要知道的一切資訊，程式呼叫 getX() 與 getY() 方法從 MouseEvent 取得滑鼠目前位置的 x 與 y 坐標值，接著將數值分別儲存到 messageX 與 messageY 實體變數供其他地方使用。

事件模型的美在於程式只需要處理想要處理的事件種類就行了，如果不在意鍵盤事件，就不要註冊它們的傾聽者，那麼不論使用者怎麼敲打鍵盤都對程式沒有任何影響。如果某個種類的事件沒有任何收聽者，Java 就不會產生該類型的事件，使得事件處理機制變得十分有效率 [2]。

在討論事件的同時，應該要提到偷渡進 HelloJava2 的小變動：

```
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

這行程式告訴視窗在使用者按下關閉視窗鈕的時候要結束應用程式，如同絕大多數的 GUI 互動，這個「預設」的關閉動作也是用事件控制，也可以註冊視窗傾聽器，在使用者按下關閉鈕的時候收到通知，再採取需要的行動，但使用這個便利的方法就能夠處理一般常見的情況。

2 Java 1.0 的事件處理機制就完全是另一回事了，早期的 Java 並沒有事件傾聽器的概念，所有的事件處理都必須透過覆寫基礎 GUI 類別的方法達成，這種做法非常沒有效率，也會導致不良的設計以及許多特殊用途的元件。

碁峰 www.gotop.com.tw

最後，以上的討論迴避了一些問題：系統是怎麼知道類別裡有需要的 mouseDragged() 與 mouseMoved() 方法（這些名稱又是從哪裡來的）？為什麼一定要定義一個什麼事也沒有做的 mouseMoved() 方法？這些問題的答案都與介面（interface）有關，在討論完最後一部分的 repaint() 後就會介紹介面了。

repaint() 方法

由於程式（在使用者拉動滑鼠時）會改變訊息的坐標，因此需要讓 HelloComponent2 重新繪製自己，這可以透過呼叫 repaint() 達成，這個方法會要求系統在某個時間重繪螢幕，程式無法直接呼叫 paintComponent()，就算想要這麼做，也沒有需要傳給 paintComponent() 的 graphics context。

程式可以使用 JComponent 類別的 repaint() 方法要求重繪元件，repaint() 方法會讓 Java 的視窗系統在下一個許可的時間安排呼叫元件的 paintComponent() 方法，Java 會在呼叫時提供必要的 Graphics 物件，如圖 2-16。

這種運作模式並不單單只是因為缺少必要的 graphics context 所造成的不便，這種運作模式最主要的好處在於重繪是由其他人處理，程式可以繼續處理自己的邏輯，Java 系統另外有個獨立執行的執行緒，專供處理所有的 repaint() 請求，這個執行緒能夠依需要排程與協調 repaint() 請求，避免捲動視窗等需要大量重繪的情境造成視窗系統過載。另一個好處是所有的描繪函式都必須透過 paintComponent() 封裝，也就不會散落在程式各處。

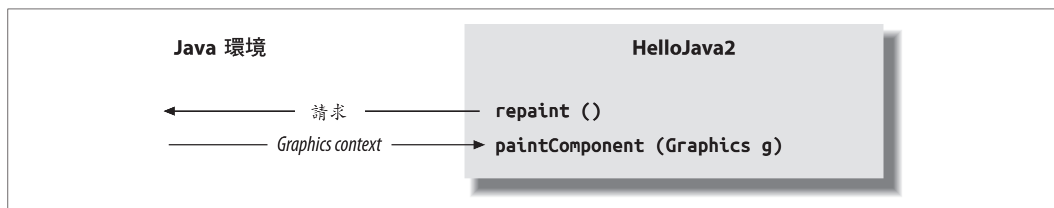

圖 2-16　呼叫 repaint() 方法

介面

現在可以回答先前避開的問題了：發生滑鼠事件時，系統是怎麼知道該呼叫 mouseDragged()？是不是因為 mouseDragged() 是個事件處理方法必須知道的特別名稱？不完全如此，這個問題的答案需要提到介面，這也是 Java 程式語言裡最重要的特性之一。

碁峰 www.gotop.com.tw

介面的第一個跡象出現在 HelloComponent2 class 的程式碼裡：程式中提到類別實作（implement）了 MouseMotionListener 介面（interface）：

```
class HelloComponent2 extends JComponent
    implements MouseMotionListener
{
```

基本上，介面是類別一定得有的方法列表，MouseMotionListener 要求類別必須要有 mouseDragged() 與 mouseMoved() 這兩個方法。介面不限定方法的行為，實際上 mouseMoved() 什麼也沒做，但介面裡的確表示了這兩個方法必須接受 MouseEvent 作為引數，同時沒有傳回值（也就是 void）。

介面是開發人員與編譯器間的約定，藉由表示類別實作了 MouseMotionListner 介面，代表類別中必須存在這些方法，可以讓系統其他部分呼叫，如果開發人員沒有提供這些方法，就會發生編譯錯誤。

介面對程式的影響不只如此，介面的行為與類別十分相似，例如，方法可以傳回 MouseMotionListener 或接受 MouseMotionListener 作為引數，當程式透過介面的名稱參照物件，就表示程式並不在乎物件實際上的類別，只要類別實作了需要的介面就行了。addMouseMotionListener() 就是這樣的方法：它的引數必須是個實作了 MouseMotionListener 介面的物件，範例程式傳入的是 this，也就是 HelloComponent2 物件自己，傳入的是 JComponent 實體這件事並不重要，傳入的可以是 Cookie、Adrdvark 或其他任何的類別，重要的是類別實作了 MouseMotionListener，也就是必須要有這兩個相同名稱的方法。這也是為什麼範例程式中必須要有個什麼事也沒做的 mouseMoved() 方法的原因，因為 MouseMotionListener 介面要求我們必須提供這個方法。

Java 包含了許多定義行為規範的介面，編譯器與類別間的合約概念非常重要，還有與先前類似的情況，程式只在意某些特定的能力，而不是實際的類別，如同滑鼠事件的傾聽器一般。介面提供了一種以物件能力操作物件的方法，讓開發人員不需要在意或知道物件實際上的型別，這是把 Java 當作物件導向式程式語言使用時一個非常重要的觀念，會在第五章更詳細的討論。

第五章也會討論到以介面跳脫 Java 「每個類別只擴展一個類別」規則的方法，Java 裡的類別只能夠擴展一個類別，但可以實作多個介面。介面可以作為資料型別，可以擴展其他介面（但不能擴展 class），也可以被類別繼承（如果 A 類別實作了 B 介面，那麼 A 的子類別也就實作了 B），最關鍵的不同在於類別實際上並沒有從介面繼承到任何方法，介面只限定了類別必須要有的方法。

碁峯 www.gotop.com.tw

離別與再相見

現在是與 `HelloJava` 道別的時候了，希望讀者已經對 Java 的一些特性以及撰寫與執行 Java 程式的方式有些感覺，這個簡單的介紹應該能幫助讀者探索其他 Java 程式設計的細節，如果因為對某些內容不清楚而耿耿於懷，重要的主題都會在後續相關章節裡再次介紹，這個入門介紹只是簡單的試車，讓讀者熟悉重要概念與術語，在下次再遇到的時候能夠更快的上手。

現在要拋開 `HelloJava` 了，在下一章會介紹 Java 世界的工具，讀者會看到 *javac* 等已經用過的命令的其他細節，也會看到其他重要的工具。請繼續讀下去，準備與 Java 開發人員的幾個新好友打招呼吧！

碁峰
www.gotop.com.tw